14

Managing MySQL Security

An important component of administering any database is ensuring that only those users that you want to be able to access the database can do so, while preventing access by all other users. Not only should you be able to control who can log on to the MySQL server, but you should be able to determine what actions authenticated users can take once they connect to the server. All RDBMS products support some level of security in order to protect the data stored in their systems' databases — and MySQL is no exception.

When a user logs on to a MySQL server, MySQL permits the user to perform only approved operations. MySQL security is managed through a set of tables and privileges that determine who can establish a connection to the MySQL server, from what host that connection can be established, and what actions the user (from the specified host) can take. In this chapter, you learn how this system is set up and how you can add user accounts or remove them from the tables. You also learn how to permit users to perform certain actions, while preventing them from taking other actions. To facilitate your ability to configure MySQL security, this chapter provides information about the following topics:

- The MySQL grant tables, including the user, db, host, tables_priv, and columns_priv tables
- The process used to authenticate connections to the MySQL server and to verify the privileges necessary to perform various operations
- The statements necessary to manage MySQL user accounts, including the `GRANT`, `SHOW GRANTS`, `SET PASSWORD`, `FLUSH PRIVILEGES`, `REVOKE`, and `DROP USER` statements

The Access Privilege System

In Chapter 3, you were introduced to the mysql database created by default when you install MySQL. As you learned in that chapter, the mysql database is an administrative database that contains tables related to securing the MySQL installation, storing user-defined functions, and providing data related to the MySQL help system and to time-zone functionality. Of particular concern to preventing unauthorized access to the MySQL server are the security-related tables, which are referred to as the *grant tables*. The grant tables are a set of five tables in the mysql database used to control access to the MySQL server and to the databases managed by that server. The grant tables define which users can

access MySQL, from which computers that access is supported, what actions those users can perform, and on which database components those actions can be performed. For example, the grant tables allow you to specify which users can view data in a particular table and which users can actually update that data. In this section, you learn about each of the five grant tables and how those tables are used to authenticate users and determine what operations they can perform.

> *Working with the grant tables and MySQL security is only one aspect of securing a MySQL installation. You should also ensure the security of the MySQL-related files and the network used to access data in the MySQL databases. A discussion about security specific to your operating system or your network is beyond the scope of this book. For information about system and network security, consult the appropriate product and system documentation.*

MySQL Grant Tables

When you install MySQL, five grant tables are added to the mysql database. These tables — user, db, host, tables_priv, and columns_priv — each perform a specific role in either authenticating users or determining whether a particular action can be carried out. Each table contains two types of columns:

- **Scope columns:** Columns in a grant table that determine who has access to the MySQL server and the extent, or scope, of that access. Depending on the grant table, the scope columns can include the user account name, the host from which that user connects, the user account's password, or, when appropriate, the name of a specific database, table, or column.
- **Privilege columns:** Columns in a grant table that determine what operations can be performed by the user identified in the scope columns. For most grant tables, the privilege columns define the level of access a user account has to the data and the extent to which the user can manipulate that data or allow others to access and manipulate that data. The user table also includes privilege columns that permit administrative operations, require encrypted connections, and specify limits on connections. The default value for most privilege columns is N (for *no,* or *false*). When a privilege is assigned to a user, the value in the related privilege column is set to Y (for *yes,* or *true*).

Through the use of privileges, all five tables participate in the process of determining whether a user can perform a particular operation. (An operation refers to an event such as issuing a `SELECT` statement to view data in a table or a `CREATE TABLE` statement to create a table.) Later in the chapter, in the section MySQL Privileges, you learn about the different types of privileges and how they're supported by the grant tables. After that (in the section MySQL Access Control), you learn how the grant tables and privileges are used to authenticate connections to the MySQL server and to permit various types of operations. First, though, you take a closer look at each grant table.

The user Table

The user table is the primary grant table in the mysql database. The table controls who can connect to MySQL, from which hosts they can connect, and what global privileges they have. A global privilege is one that applies to the MySQL server or to any database in the system. For example, a global privilege might be used to allow a user to view data in every table in every database or to allow the user to display a list of the current processes running.

In MySQL, a user is identified not only by the user account name, but also by the host from which the user connects. For example, `user1@domain1.com` *is considered a different user from* `user1@domain2.com`*. MySQL uses this method so that users with the same name but from different domains are not treated as the same user. When you come across a reference to a user in MySQL, that reference is usually referring to that user in association with a particular host (or with any host, if that is how the user account is set up). As you progress through this chapter, you learn how this user/host association is used in authenticating a user and in authorizing certain operations.*

As with the other grant tables, the user table contains scope columns and privilege columns. Unlike the other grant tables, though, the user table is the only table that contains different types of privilege columns. The following list describes the different types of columns in the user table:

- **Scope columns:** Includes the Host, User, and Password columns. When a connection is initiated, the connection must be made from the host specified in the Host column. In addition, the user account name used for the connection must match the value in the User column, and the password provided when the connection is initiated must match the value in the Password column. A connection is permitted only if all three values match.
- **Data-related privilege columns:** Includes those privilege columns that permit data-related operations that are global in scope. There are 11 data-related privilege columns. A privilege granted at this level applies to all tables in all databases. There are also two additional data-related privilege columns that are not currently supported, but they should be supported in later releases of MySQL.
- **Administrative privilege columns:** Includes those privilege columns that permit administrative operations to the MySQL server. There are eight administrative privilege columns.
- **Encryption-related privilege columns:** Includes the ssl_type, ssl_cipher, x509_issuer, and x509_subject columns, which define whether a user account requires a secure connection and define the nature of that connection.
- **Connection-related privilege columns:** Includes the max_questions, max_updates, and max_connections columns, which define whether a limit should be placed on the number of queries, the number of data updates, and the number of connections that can be made in an hour.

To give you an idea of how the user table is configured, assume that you have a user account named user1 that can connect from the `domain1.com` domain. You can execute a `SELECT` statement similar to the following to view how that user is listed in the user table:

```
SELECT Host, User, Select_priv, Process_priv, ssl_type, max_updates
FROM user
WHERE User='user1';
```

The `SELECT` statement retrieves values from two of the scope columns (Host and User) and several of the privilege columns. The Select_priv column assigns a data-related privilege, the Process_priv column assigns an administrative privilege, the ssl_type assigns an encryption privilege, and the max_updates column assigns a connection-related privilege. (You learn more about these privileges later in the chapter.) When you execute this statement, you receive results similar to the following:

```
+-------------+-------+-------------+--------------+----------+-------------+
| Host        | User  | Select_priv | Process_priv | ssl_type | max_updates |
+-------------+-------+-------------+--------------+----------+-------------+
| domain1.com | user1 | N           | N            | ANY      |           0 |
+-------------+-------+-------------+--------------+----------+-------------+
1 row in set (0.01 sec)
```

There are, of course, many more privileges than are shown here, but this should give you an idea of how MySQL stores user account information in the user table. As you can see, the host from which the user should connect, the name of the user account, and the columns for each privilege that can be applied globally are listed.

Every MySQL user is listed in the user table, whether or not he or she is assigned privileges in that table or any other grant table. The user table provides the widest scope in a MySQL implementation, followed by the db and host tables. If a user is not listed in the user table, that user cannot connect to MySQL.

The db Table

The purpose of the db table is to assign database-specific privileges to users. Any privileges applied in the db table are specific to the specified database. If privileges are assigned to a user for multiple databases, a row is added to the db table for each database, and privileges are then assigned for that row. (If the privileges should apply to all databases, the privileges are assigned in the user table.)

As with the other grant tables, the db table includes scope columns and privilege columns. The following list provides an overview of these columns:

- **Scope columns:** Includes the Host, Db, and User columns. For the privileges in this table to apply, the connection must be made from the host specified in the Host column, and the user account name used for the connection must match the value in the User column. If the host column is blank, then the privileges also defined in the host table are applied. Any privileges assigned in the db table apply only to the database specified in the Db column.
- **Privilege columns:** Includes those privileges that can be applied at the database level. These are the 11 data-related privileges (the same as those you see in the user table) used to permit data-related operations. There is an additional data-related privilege column that is not currently supported, but it should be supported in later releases of MySQL.

To get a better sense of the db table, assume that the user1 account has been assigned privileges specifically on the test database. You can then use a `SELECT` statement to view information about that user account:

```
SELECT Host, Db, User, Select_priv, Update_priv
FROM db
WHERE User='user1';
```

The statement retrieves information from the three scope columns (Host, Db, and User) and from two of the privilege columns (Select_priv and Update_priv), as shown in the following results:

```
+-------------+------+-------+-------------+-------------+
| Host        | Db   | User  | Select_priv | Update_priv |
+-------------+------+-------+-------------+-------------+
| domain1.com | test | user1 | Y           | N           |
+-------------+------+-------+-------------+-------------+
1 row in set (0.00 sec)
```

As you can see, the Host value is `domain1.com`, the Db value is `test`, and the User value is `User1`. As a result, the privileges assigned in this row are applied to the user1 account when that account connects from `domain1.com`. In addition, the privileges apply only to the test database. You can, of course, retrieve more privilege columns than those shown here. If a privilege is assigned to the account, a value of Y is displayed in the columns; otherwise, a value of N is displayed.

The db table works in conjunction with the host table. If the Host column in the db table is blank, MySQL checks the host table to determine whether any privileges apply to a specific database from a specific host.

The host Table

The host table is associated with the db table and is checked only when a user is listed in the db table but the Host column is blank. The combination of these two tables allows you to apply privileges to a user who connects from multiple hosts. For example, if a user named SarahW connects to MySQL from `big.domain1.com` and `little.domain1.com`, you can add the user to the db table, with a blank host value, and then add the two hosts to the host table, specifying the same database name in the SarahW row of the db table and in the `big.domain1.com` and `little.domain1.com` rows of the host table. When MySQL sees the blank host value in the db table, it looks in the host table for a Host value that matches the hostname of the connection. If there's a match, the privileges from the db table and the host table are compared to determine whether an operation is permitted. The matching privileges in both tables must be set to Y for the operation to be permitted. (You learn more about this process in the section "MySQL Access Control" later in the chapter.)

The host table, as with other grant tables, includes scope columns and privilege columns. Because the host table works in conjunction with the db table, it is the only grant table that does not include a User column. The following list describes the columns in the db table:

- **Scope columns:** Includes the Host and Db columns. For the privileges in this table to apply, the Host column in the db table must be blank, and the connection must be made from the host specified in the Host column of the host table. Any privileges assigned in the host table apply only to the database specified in the Db column of the host table.
- **Privilege columns:** Includes those privileges that can be applied at the database level for user accounts accessing the databases from specific hosts. These are the same 11 data-related privilege columns that you find in the user and db tables. The privileges are combined with the applicable privileges in the db table to permit authorized operations. There is an additional data-related privilege column that is not currently supported, but it should be supported in later releases of MySQL.

To better understand how user accounts are added to the host table, suppose that you want to allow user1 to connect from the `host host1.domain1.com` or from `host2.domain1.com` domains. You add a row for each host in the host table. To then view the rows added to the table, you can use a `SELECT` statement similar to the following:

```
SELECT Host, Db, Select_priv, Update_priv
FROM host
WHERE Host='host1.domain1.com' OR Host='host2.domain1.com';
```

As you can see, the SELECT statement retrieves values from the two scope columns (Host and Db) and two of the privilege columns, as shown in the following results:

```
+-------------------+------+-------------+-------------+
| Host              | Db   | Select_priv | Update_priv |
+-------------------+------+-------------+-------------+
| host1.domain1.com | test | Y           | N           |
| host2.domain1.com | test | Y           | N           |
+-------------------+------+-------------+-------------+
2 rows in set (0.01 sec)
```

The results include only the Host and Db scope columns, but not a User scope column because one isn't included in the host table. The host table contains the same 11 data-related privilege columns that are in the db and user tables (12 data-related columns if you include the unsupported column).

The host table is different from other grant tables in that user account information and privilege settings are not configured in the same way as the other tables. Later in the chapter you learn that the GRANT statement is the primary method that you should use to add users to your system and to assign privileges. At the same time, the REVOKE statement is the primary method that you should use to revoke privileges. The host table, however, is not affected by the GRANT statement or the REVOKE statement, which means that, if you plan to use the host table, you must set up the privileges manually, which is a less efficient method to use than the GRANT and REVOKE statements and which is more prone to errors.

This problem can be exacerbated if multiple users connect from the same hosts. You must ensure that the entries in the host table can be applied to all users in such a way that, when those privileges are compared with the privileges in the db table, each user can perform the necessary operations, without being able to perform unauthorized operations. As a result of these issues, few MySQL implementations use the host table.

The tables_priv Table

The tables_priv table is specific to table-level privileges. Any privileges assigned in this table apply only to the table specified in the tables_priv table. The following list describes the columns in the tables_priv table:

- **Scope columns:** Includes the Host, Db, User, and Table_name columns. For the privileges in this table to apply, the connection must be made from the host specified in the Host column, and the user account name used for the connection must match the value in the User column. Any privileges assigned in the tables_priv table apply only to the table specified in the Table_name column, as it exists in the database specified in the Db column.
- **Privilege columns:** Includes the Table_priv column and the Column_priv column. The Table_priv column defines the privileges applied at the table level. The Column_priv column defines the privileges applied at the column level.

Note that the tables_priv table also includes the Grantor and Timestamp columns, but they are currently unused. It is assumed that future releases of MySQL will make use of these columns.

To better understand how the tables_priv table stores user account data, assume that user1 has been assigned table- and column-level privileges. You can use a SELECT statement similar to the following to retrieve data from the tables_priv table about that user:

```
SELECT Host, Db, User, Table_name, Table_priv, Column_priv
FROM tables_priv
WHERE User='user1';
```

The statement includes the four scope columns (Host, Db, User, and Table_name) and the two privilege columns (Table_priv and Column_priv). As you learn in the next section, the privilege columns in the tables_priv and columns_priv tables work differently from the privilege columns in the other grant tables. When you execute the `SELECT` statement, your results should be similar to the following:

```
+-------------+------+-------+------------+------------+-------------+
| Host        | Db   | User  | Table_name | Table_priv | Column_priv |
+-------------+------+-------+------------+------------+-------------+
| domain1.com | test | user1 | Books      | Select     | Update      |
+-------------+------+-------+------------+------------+-------------+
1 row in set (0.00 sec)
```

As you can see, the scope columns include not only the host, user, and database (as you saw in the db table) but also the name of the table. As a result, any privileges assigned to this user account in this table are specifically for the Books table.

The tables_priv table works in conjunction with the columns_priv table. If the Column_priv column in the tables_priv table contains a value, MySQL checks the columns_priv table for specifics about the privileges that apply to the individual columns.

The columns_priv Table

The columns_priv table shows the privileges associated with individual columns. When a user is listed in the tables_priv table and the columns_priv table, the privileges specified in the Column_priv column of the columns_priv table are applied to the columns, and the privileges specified in the Table_priv column of the tables_priv table are applied to the table as a whole. The following list describes the columns in the columns_priv table:

- **Scope columns:** Includes the Host, Db, User, Table_name, and Column_name columns. For the privileges in this table to apply, the connection must be made from the host specified in the Host column, and the user account name used for the connection must match the value in the User column. Any privileges assigned in the columns_priv table apply only to the column specified in the Column_name column, as it exists in the table specified in the Table_name column, which exists in the database specified in the Db column.
- **Privilege columns:** Includes the Column_priv column, which defines the privileges applied at the column level.

Note that the columns_priv table also includes the Timestamp column, but is currently unused. It is assumed that future releases of MySQL will make use of this column.

In the previous example, you saw how the user1 account appears in the tables_priv table. Now you can use a `SELECT` statement to retrieve information about that account in the columns_priv table:

```
SELECT Host, Db, User, Table_name, Column_name, Column_priv
FROM columns_priv
WHERE User='user1';
```

As you can see, the statement retrieves data from the scope columns (Host, Db, User, Table_name, and Column_name) and the privilege column (Column_priv), as shown in the following results:

```
+-------------+------+-------+------------+-------------+-------------+
| Host        | Db   | User  | Table_name | Column_name | Column_priv |
+-------------+------+-------+------------+-------------+-------------+
| domain1.com | test | user1 | books      | BookTitle   | Update      |
| domain1.com | test | user1 | books      | Copyright   | Update      |
+-------------+------+-------+------------+-------------+-------------+
2 rows in set (0.00 sec)
```

Each row in the columns_priv table assigns privileges to a specific column. The columns_priv table is the most granular of all the tables, in terms of assigning privileges to users. When MySQL authorizes a user to perform an operation, however, all applicable grant tables are examined, starting with the user table and working down to the columns_priv table. For example, if a user is granted select privileges on a table, but update privileges on only one column in the table, the user can still retrieve data from the entire table, but update data only in that one column. The following section takes a closer look at the different privileges supported by MySQL.

MySQL Privileges

The user, db, and host tables contain columns that each represent individual privileges. All three tables include the data-related privileges, which deal specifically with managing data. As you saw previously, only the user table contains privileges that are specific to administration and connectivity. The following table describes the privileges that can be assigned in the user, db, and host tables, and it shows which tables contain which privileges.

Column	Type	Allows user to	user table	db table	host table
Select_priv	Data-related	Query data in a database.	X	X	X
Insert_priv	Data-related	Insert data in a database.	X	X	X
Update_priv	Data-related	Update data in a database.	X	X	X
Delete_priv	Data-related	Delete data from a database.	X	X	X
Create_priv	Data-related	Create a table in a database.	X	X	X
Drop_priv	Data-related	Remove a table from a database.	X	X	X
Reload_priv	Administrative	Reload the data in the grant tables in MySQL.	X		

Column	Type	Allows user to	user table	db table	host table
Shutdown_priv	Administrative	Shut down the MySQL server.	X		
Process_priv	Administrative	View a list of MySQL processes.	X		
File_priv	Administrative	Export data from a database into a file.	X		
Grant_priv	Data-related	Grant privileges on database objects.	X	X	X
Index_priv	Data-related	Create and delete indexes in a database.	X	X	X
Alter_priv	Data-related	Alter database objects.	X	X	X
Show_db_priv	Administrative	View all databases.	X		
Super_priv	Administrative	Perform advanced administrative tasks.	X		
Create_tmp_table_priv	Data-related	Create temporary tables.	X	X	X
Lock_tables_priv	Data-related	Place locks on tables.	X	X	X
Repl_slave_priv	Administrative	Read binary logs for a replication master.	X		
Repl_client_priv	Administrative	Request information about slave and master servers used for replication.	X		
ssl_type	Encryption-related	Specifies whether a secure connection is required. If required, the column specifies the type of secure connection.	X		
ssl_cipher	Encryption-related	Specifies the cipher method that should be used for a connection. If the column is blank, no special cipher method is required.	X		

Table continued on following page

Column	Type	Allows user to	user table	db table	host table
x509_issuer	Encryption-related	Specifies the name of the certificate authority that issues the x509 certificate. The name should be used for an x509 connection. If the column is blank, the issuer name is not required.	X		
x509_subject	Encryption-related	Specifies the subject that should be included on the x509 certificate when establishing a secure connection. If the column is blank, the subject is not required.	X		
max_questions	Connection-related	Specifies the number of queries that an account can issue in an hour. If set to 0, the user account can issue an unlimited number of queries.	X		
max_updates	Connection-related	Specifies the number of data updates that an account can perform in an hour. If set to 0, the user account can perform an unlimited number of updates.	X		
max_connections	Connection-related	Specifies the number of connections that an account can establish in an hour. If set to 0, the user account can connect an unlimited number of times.	X		

The privileges are listed in this table in the order they appear in the user table. Because permissions in the user table are applied globally, it is the only table that contains all privileges. The more granular the privileges, the fewer privileges there are, which is reflected in the grant tables. For example, the user table has more privilege columns than the db table, which is more granular than the user table.

Some of the grant tables also include the Execute_priv and the References_priv, neither of which are currently supported. It appears that the Execute_priv will be related to stored procedures when they're implemented in MySQL, and the References_priv will be related to foreign keys.

All the data-related and administrative privilege columns in the user, db, and host tables are configured with the ENUM data type and assigned the values N or Y, with N being the default. When a permission is granted to an account, the value is changed to Y. For the encryption-related and connection-related columns, the data types vary, depending on the type of data that can be inserted in those columns. In the "Managing MySQL User Accounts" section later in the chapter, you learn how to use the GRANT statement to insert values in these columns.

Privileges are assigned differently in the tables_priv and columns_priv tables than they're assigned in the user, db, and host tables. The tables_priv table includes the Table_priv and Column_priv columns, and the columns_priv table includes only the Column_priv column. Each of these columns is configured with the SET data type, which means that a set of values is defined for each column. One or more of those values can be inserted in the columns. The following table lists the values included with each column.

Column	Privilege values
Table_priv	Select, Insert, Update, Delete, Create, Drop, Grant, Index, Alter
Column_priv	Select, Insert, Update

Both columns also include a References privilege, but that privilege currently is not used by MySQL.

The values in the Table_priv and Column_priv columns have their counterparts with some of the data-related privilege columns you find in the user, db, and host tables. The names of the values clearly indicate to which privileges they are related. For example, the Select value has its counterpart with the Select_priv column in the user, db, and host tables. In both cases, the privilege allows the specified user to retrieve data from a database.

MySQL Access Control

When MySQL permits a user to conduct various operations in the MySQL environment, it first authenticates those connections to allow access to the MySQL server, and then it verifies the privileges assigned to that user account to determine whether the requested operations are permitted.

Authenticating Connections

The first step in allowing a user to access the MySQL server is to ensure that the user has that access. If access is permitted, MySQL authenticates the connection. The connection is authenticated only if the Host value in the user table matches the name of the host from which the connection is being established. In addition, the username used for the connection must match the value in the User column, and if a password is required, the password supplied for the connection must match the value in the Password column. If the parameters supplied by the connection match all the applicable values in the user table, the connection is permitted.

If a Host value in the user table contains the percentage (%) wildcard, the user can connect from any host. If the User column is blank, any user can connect to the server from the specified host. (These types of users are referred to as anonymous users.) If the Host value is the percentage wildcard and the User column is blank, any user from any host can connect to the server. The Password column can also be blank.

This does not mean, though, that any password is acceptable. It means that the user must supply a blank password. For example, if the user specifies the `-p` option when launching MySQL and then is prompted for a password, the user should simply press Enter and not enter any password.

If you expect to allow anonymous users or plan to allow users to connect from any host, you must plan your user table carefully to ensure that the right user is associated with the correct privileges. To plan your user accounts, you should keep in mind how MySQL accesses the user table:

1. When the MySQL server starts, data from the user table is copied to memory in sorted order.
2. When a client attempts to log on to the server, the user account is checked against the sorted user data in memory.
3. The server uses the first applicable entry to authenticate a user, based first on the Host value and then on the User value.

MySQL first sorts the user table according to the Host column and then according to the User column. For the Host column, specific values are listed first, followed by less specific values, such as those that use the percentage wildcard. Rows that have the same value in the Host column are then sorted according to the value in the User column, again, with the specific values listed first and the least specific (a blank) last.

This sorting is important because MySQL, when authenticating a user, first checks the Host column for a match and then checks the User column. Whichever row in the user table provides the first match, that is the user account used to authenticate the user and subsequently assign privileges to that user. If there are no wildcards in the host column and no blanks in the User column, MySQL simply matches the host-name and username to the Host and User values and authenticates the user. The use of wildcards and blanks in the Host and User column, however, can result in a user being incorrectly authenticated.

To help illustrate the implications of the user table sorting, take a look at the following result set, which shows the Host and User values from the user table in a MySQL installation:

```
+-------------+-------+
| Host        | User  |
+-------------+-------+
| %           | root  |
| domain1.com | user1 |
| localhost   |       |
| %           |       |
+-------------+-------+
4 row in set (0.01 sec)
```

As you saw earlier, when the user table is copied to memory, the data is sorted according to the Host and User columns, as shown in the following result set:

```
+-------------+-------+
| Host        | User  |
+-------------+-------+
| domain1.com | user1 |
| localhost   |       |
| %           | root  |
| %           |       |
+-------------+-------+
4 row in set (0.01 sec)
```

Notice that the rows that contain the percentage wildcard in the Host column are the last rows. The percentage represents the least specific type of host, so it appears last. Now suppose that the root user attempts to log in from the local computer. According to the user table, there is only one entry for the root account: `root@%`. In theory, this would mean that the root account can log in from any host, including the local computer. When the user logs on, MySQL first checks the Host column and finds the first match, which is the localhost value. MySQL then checks the User column and finds only a blank value, indicating that anonymous users are permitted, which is also a match. As a result, the root account is logged on as an anonymous user, rather than the primary administrator. This means that the root user is now operating under the privileges granted to the anonymous user rather than to the root user.

One way to get around this situation is to create two entries in the user table for a specified user, as is done for the root user when you install MySQL. In Windows, the root user can log on as `root@localhost` or as `root@%`, and in Linux, the root user can log on as `root@localhost` or as `root@<host>`, where *<host>* is the name of the local computer. This way, even if the percentage wildcard is used for the Host value or a blank is used for the User value, MySQL can still identify the root account. By including an entry specifically for localhost, anonymous users can be permitted, but the specified accounts are still protected. As a result, when the root user logs on, there are now two rows for localhost. These two rows are then sorted first by the root user and then by the anonymous users. A blank value is always at the end of the list. The root user is then logged on with the correct privileges.

> *If anonymous accounts are allowed to access the MySQL server, normally passwords would not be permitted. This can even further complicate the user who is inadvertently treated as an anonymous user. For example, if a user tries to log on as root and supplies a password, but MySQL treats that user as anonymous, the password is interpreted as an incorrect value because MySQL is expecting a blank password. As a result, the connection is denied.*

By ensuring that the user table has the necessary entries for each user, users can be correctly logged on to the MySQL server. The operations that the user can perform are still limited to those permitted by the privileges associated with that user account.

Verifying Privileges

After MySQL authenticates a user, it checks the privileges associated with that user account to determine what operations the user can perform. When verifying privileges, MySQL first checks the user table. If privileges have not been granted to the account in the user table, MySQL checks the db table and, if appropriate, the host table. If privileges have not been granted at the database level, MySQL checks the tables_priv table, and, if appropriate, the columns_priv table. If, after checking all applicable tables, the operation is not permitted, the operation fails.

> *MySQL drills down through the grant tables (from the user table down to the columns_priv table) only as far as necessary. For example, if a user logs on to the MySQL server and tries to perform an administrative action, MySQL checks only the user table because only that table contains administrative privileges.*

Figure 14-1 illustrates the process that MySQL uses to authorize users to perform the requested operation. The figures show the process only as far as it goes to checking the tables_priv and columns_priv tables. (Figure 14-2 covers those two tables.) Figure 14-1 assumes that the requested operation is one that could potentially require permissions as far down as those that can be assigned in the tables_priv and columns_priv tables. Of course, if any operation does not require that MySQL drill down through the grant tables that far, the operation either is permitted or fails at whatever point MySQL no longer needs to continue checking tables.

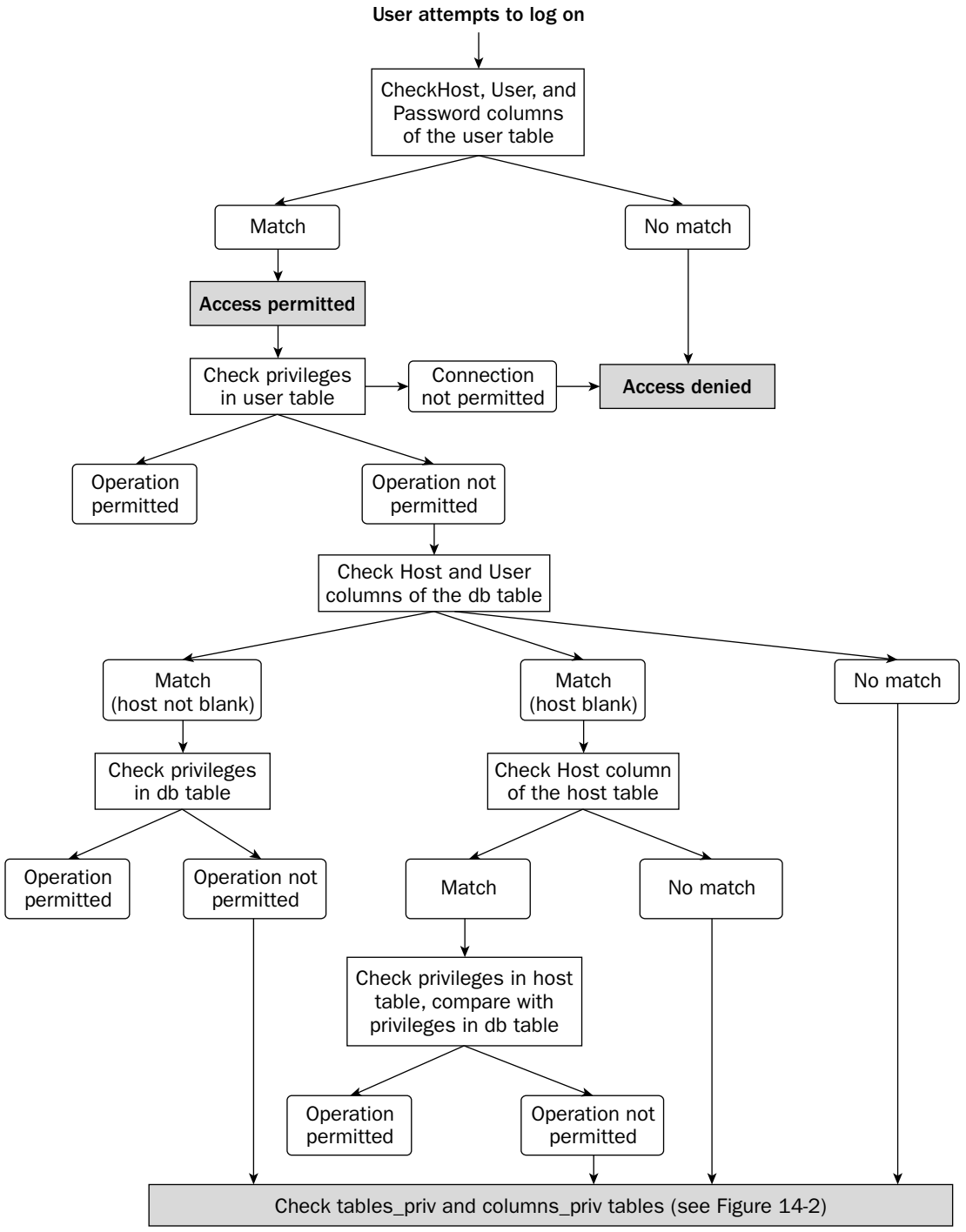

Figure 14-1

The following steps describe the process that MySQL follows when verifying privileges:

1. The user or the application initiates a connection, supplying a hostname, username, and password, if a password has been assigned to the user account. For users or applications attempting to log on anonymously, only the hostname is provided.
2. If no match is found in the Host, User, and Password columns of the user table, access is denied. If a match is found, access to the MySQL server is permitted.
3. MySQL then checks the privileges in the user table. If the connection exceeds the connections-per-hour limit, access is denied. If the connection does not meet any encryption-related privileges assigned to the connection, access is denied. Otherwise, the operation being requested by the connection is compared to the data-related and administrative privileges defined in the user table. For example, if the connection attempts to execute a `SELECT` statement, the `SELECT` privilege is checked to see whether a `SELECT` statement operation is permitted.
4. If the privileges permit the operation, MySQL carries out that operation. If the privileges do not permit the operation, MySQL checks the db table:
    - If the connection hostname and username match the values in the Host and User columns of the db table, MySQL checks the privileges in that table. If the privileges permit the operation for the specified database, MySQL carries out that operation. If the privileges do not permit the operation, MySQL checks the tables_priv and columns_priv tables. (See Figure 14-2.)
    - If the username matches the value in the User column of the db table and the Host column for that table is blank, MySQL checks the Host column of the host table. If the connection hostname matches the value in the Host column, MySQL checks the privileges in the host table and compares them to the privileges in the db table. If the privileges permit the operation for the specified database, MySQL carries out that operation. If the privileges do not permit the operation, MySQL checks the tables_priv and columns_priv tables. If the connection hostname does not match the value in the Host column of the host table, MySQL checks the tables_priv and columns_priv tables.
    - If the connection hostname and username do not match the values in the Host and User columns of the db table, MySQL checks the tables_priv and columns_priv tables.

If, after checking the db and host tables, MySQL reaches a point when it still cannot determine whether to permit the operation, it checks the tables_priv table and, if necessary, the columns_priv table. Figure 14-2 illustrates the process that MySQL uses to check these two tables. This process is a continuation of the privilege checking shown in Figure 14-1.

MySQL first checks to see whether the connection hostname and username are listed in the Host and User columns of the tables_priv table. If there is a match, MySQL checks the privileges in that table. If there is no match, the operation fails.

If MySQL does match the hostname and username to the values in the tables_priv table, MySQL takes one of the following steps:

- If the privileges in the Table_priv column of the tables_priv table permit the operation, MySQL carries out the operation.

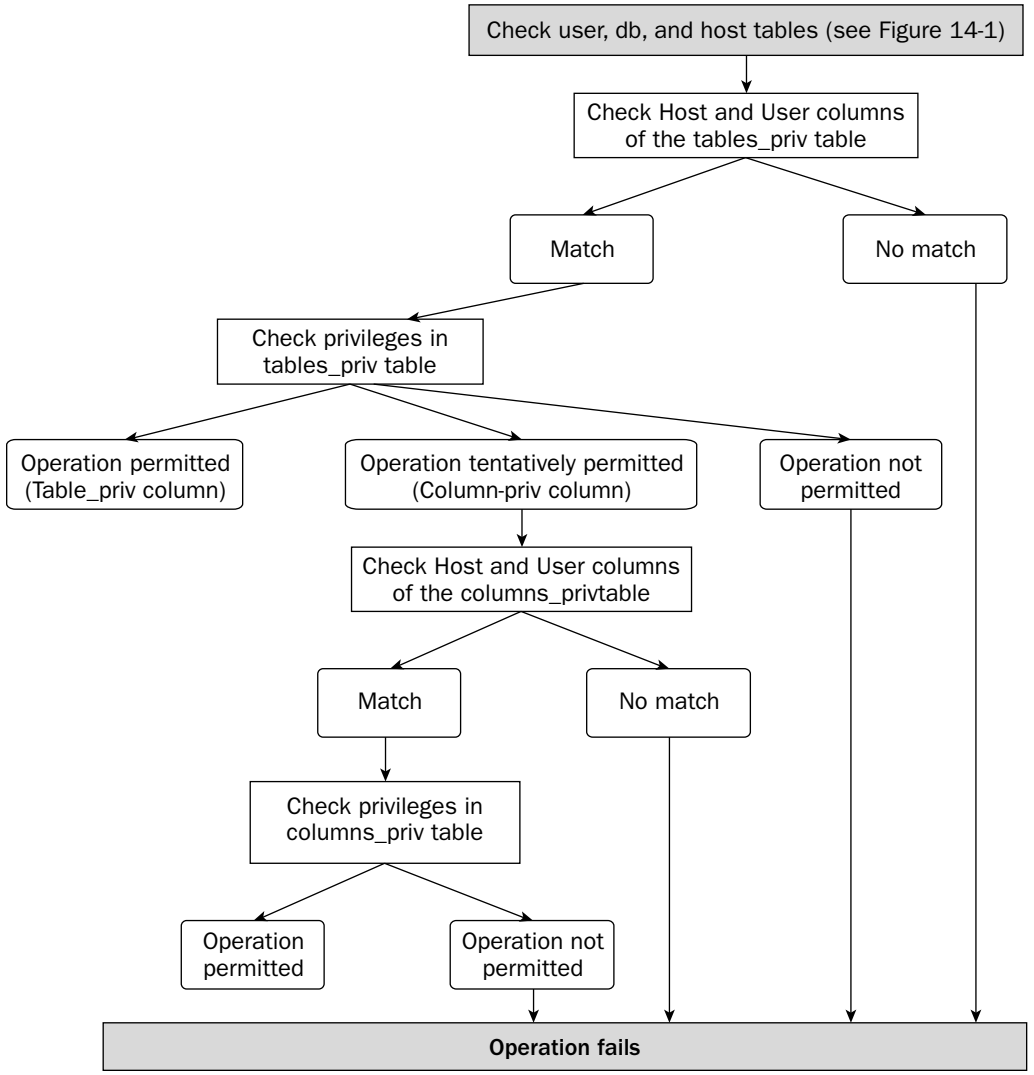

Figure 14-2

- If the privileges in the Column_priv column of the tables_priv table permit the operation, MySQL checks the columns_priv table:
  - If the hostname and username match the values in the Host and User columns of the columns_priv table, MySQL checks the privileges in the Column_priv column of the columns_priv table. If the privileges permit the operation, MySQL carries out the operation. If the privileges do not permit the operation, the operation fails.
  - If the hostname and username do not match the values in the Host and User columns of the columns_priv table, the operation fails.
- If the privileges in the tables_priv table don't permit the operation, the operation fails.

As you can see, there are many places in the authorization process at which an operation can succeed or fail. As a result, it's important that you set up your user accounts with great care to ensure that those users who should have access can log on to the server and perform the necessary operations and that those users who should not have access are prevented from logging on to the MySQL server or viewing or modifying data in the MySQL databases.

Managing MySQL User Accounts

MySQL provides a number of SQL statements that allow you to manage your user accounts. By using these statements, you can add user accounts to your system and grant privileges to those accounts. You can also view the privileges that have been assigned to a user account, or you can change the password for that account. In addition, the statements allow you to revoke privileges and drop users from your system. In this section, you learn about these SQL statements and how they can be used to secure your MySQL installation effectively.

Adding Users and Granting Privileges

You can add a user account to your system and assign privileges to that account by using the `GRANT` statement, which allows you to perform both operations in a single statement. Although you can insert account information directly in the grant tables, the `GRANT` statement is easier and is less likely to result in errors that can occur if information is added in different grant tables that conflict with each other. Once you have added a user account and assigned privileges to that account, you can use the `SHOW GRANTS` statement to view details about that account.

Using the GRANT Statement

Now that you have an overview of how the grant tables are used to authenticate users and associate privileges with those users, you're ready to look at how you actually create a user account and assign privileges to the account. To perform both operations, you should use the `GRANT` statement, which is shown in the following syntax:

```
GRANT <privilege> [(<column> [{, <column>}...])]
   [{, <privilege> [(<column> [{, <column>}...])]}...]
ON {<table> | * | *.* | <database>.*}
TO '<user>'@'<host>' [IDENTIFIED BY [PASSWORD] '<new password>']
   [{, '<user>'@'<host>' [IDENTIFIED BY [PASSWORD] '<new password>']}...]
[REQUIRE {NONE | SSL | X509 | {<require definition>}]
[WITH <with option> [<with option>...]]

<require definition>::=
<require option> [[AND] <require option>] [[AND] <require option>]

<require option>::=
{CIPHER '<string>'}
| {ISSUER '<string>'}
| {SUBJECT '<string>'}

<with option>::=
{GRANT OPTION}
| {MAX_QUERIES_PER_HOUR <count>}
| {MAX_UPDATES_PER_HOUR <count>}
| {MAX_CONNECTIONS_PER_HOUR <count>}
```

As with other SQL statements, the GRANT statement is made up of multiple clauses and options, some of which are optional and some of which are required. As you're learning about each of these clauses and you start creating your own GRANT statements, you should keep in mind that the MySQL server treats some scope column values as case sensitive (whether or not they're treated this way by your operating system). The following table shows the case sensitivity of each scope column.

Column	Case sensitive?
Host	No
User	Yes
Password	Yes
Db	Yes
Table_name	Yes
Column_name	No

When working with the GRANT statement, you want to ensure that you enter User, Password, Db, and Table_name values exactly as you intend for them to be treated once the account is created.

When you use the GRANT statement to set up a user account, the statement adds the necessary data to the grant tables. Whenever you create an account, a row is added to the user table. Whether rows are added to other grant tables depends on the level at which the privileges are being granted (for example, a global level versus a database level). In addition to using the GRANT statement to add the necessary rows to the grant tables, you can also use INSERT statements to add the necessary data. Using INSERT statements can be far more cumbersome and prone to error. For that reason, using a GRANT statement is the recommended method for adding user accounts to your system. Now take a look at how to create the required clauses in your GRANT statement.

Defining the Mandatory Clauses of the GRANT Statement

If you refer back to the GRANT statement syntax, notice that there are three required clauses: the GRANT clause, the ON clause, and the TO clause. In this section, you learn how to define each of these clauses.

Defining the GRANT Clause

The GRANT clause determines the type of privileges that should be assigned to the user account. As the following syntax shows, you can specify one or more privileges, and you can specify one or more column names with each privilege:

```
GRANT <privilege> [(<column> [{, <column>}...])]
   [{, <privilege> [(<column> [{, <column>}...])]}...]
```

Each privilege is associated with a privilege column in the user, db, or host table or with one of the privileges listed in the Table_priv or Column_priv columns of the tables_priv and columns_priv tables. The following table describes all the privileges that you can use (in place of the <privilege> placeholder) available to the GRANT statement. The table also shows the privilege column that is associated with that

privilege. For the tables_priv and columns_priv table, the privilege name is associated with the appropriate option in the Table_priv and Column_priv columns. For example, the `INSERT` privilege is associated with the Insert column option.

Privilege syntax	Columns set to Y	Actions permitted
`ALL [PRIVILEGES]`	All columns (at the level that the `GRANT` statement applies to), except the Grant_priv column	Execute all statements except `GRANT`, `REVOKE`, and `DROP USER` statements.
`ALTER`	Alter_priv	Execute `ALTER TABLE` statements.
`CREATE`	Create_priv	Execute `CREATE TABLE` statements.
`CREATE TEMPORARY TABLES`	Create_tmp_table_priv	Execute `CREATE TEMPORARY TABLE` statements.
`DELETE`	Delete_priv	Execute `DELETE` statements.
`DROP`	Drop_priv	Execute `DROP TABLE` statements.
`FILE`	File_priv	Execute `SELECT...INTO OUTFILE` and `LOAD DATA INFILE` statements.
`INDEX`	Index_priv	Execute `CREATE INDEX` and `DROP INDEX` statements.
`INSERT`	Insert_priv	Execute `INSERT` statements.
`GRANT OPTION`	Grant_priv	Execute `GRANT`, `REVOKE`, and `DROP USER` statements.
`LOCK TABLES`	Lock_tables_priv	Execute `LOCK TABLES` statements. (User must also have `SELECT` privilege.)
`PROCESS`	Process_priv	Execute `SHOW FULL PROCESSLIST` statements.
`RELOAD`	Reload_priv	Execute `FLUSH` statements.
`REPLICATION CLIENT`	Repl_client_priv	Locate slave and master replication servers.
`REPLICATION SLAVE`	Repl_slave_priv	Read binary log events from master replication servers.
`SELECT`	Select_priv	Execute `SELECT` statements.
`SHOW DATABASES`	Show_db_priv	Execute `SHOW DATABASES` statements.
`SHUTDOWN`	Shutdown_priv	Use shutdown option of the mysqladmin client utility

Table continued on following page

Privilege syntax	Columns set to Y	Actions permitted
SUPER	Super_priv	Execute CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL statements; use the debug command of the mysqladmin client utility; and connect to MySQL server even if max_connections system variable limit has been reached.
UPDATE	Update_priv	Execute UPDATE statements.
USAGE	No columns affected	No actions permitted. Option used to add user with no privileges or to update user options not related to privileges.

You might have noticed that not all privileges are represented here. The encryption-related and connection-related privileges are assigned in different clauses, which are described later in the chapter.

When you specify privileges in the GRANT clause, you simply include the privilege name and, if applicable, the column names, enclosed in the parentheses. If you include more than one privilege, they must be separated with commas. If you include more than one column for a privilege, the column names must be separated by commas.

Defining the ON Clause

After you define the GRANT clause, you can define the ON clause, shown in the following syntax:

```
ON {<table> | * | *.* | <database>.*}
```

The ON clause specifies to which tables or database the GRANT statement applies. As you can see, the clause includes four options. The option you choose determines the level at which the privileges are applied. The following list describes how the options can be used for each level:

- **Global:** Use the double wildcard (`*.*`) option to specify that the privileges should apply to the MySQL server and to all databases and their tables. You can also use the single wildcard (`*`) option if no database is active when executing the GRANT statement; otherwise, the single wildcard option applies only to the active database.
- **Database:** Use the *<database>*`.*` option to specify that the privileges should apply to the specified database as a whole and to all tables in that database. You can also use the single wildcard (`*`) option if a database is active; otherwise, the single wildcard option applies globally.
- **Table:** Use the *<table>* option to specify that the privileges should apply only to that table. The option is usually preceded by the database name and a period. For example, the books table in the test database is referred to as test.books.
- **Column:** Use the *<table>* option to specify that the privileges should apply only to that table, as is the case when assigning table-level privileges. To make the GRANT table column-specific, the *<table>* option should be used in conjunction with the columns specified in the GRANT clause.

You can use a single GRANT statement if you want to grant privileges at both the table level and at the column level. To do so, specify column names in the GRANT clause only for those columns that should be associated with privileges at the column level. Do not specify column names for the other privileges so that those privileges are applied to the table as a whole. If you want to grant privileges that combine other levels (for example, a SELECT privilege at a global level and a DELETE privilege at a table level), you should use separate GRANT statements.

Defining the TO Clause

The final required clause that you must include in your GRANT statement is the TO clause, which is shown in the following syntax:

```
TO '<user>'@'<host>' [IDENTIFIED BY [PASSWORD] '<new password>']
   [{, '<user>'@'<host>' [IDENTIFIED BY [PASSWORD] '<new password>']}...]
```

As the syntax illustrates, you can assign privileges to one or more users, as long as you separate each pair of user definitions with a comma. For each user account, you must specify a username, hostname, and optionally a password. When assigning values in the TO clause, keep the following guidelines in mind:

- **Host:** The host from which the user will be connecting. You can specify a hostname, an IP address, or localhost. You can use the percentage (%) and underscore (_) wildcards in your hostname. The percentage wildcard indicates that any number of any character can be used. The underscore wildcard indicates that any single character can be used. You can use the percentage wildcard with no other values to indicate that any host is acceptable.
- **User:** The username associated with the user account that is being created. You cannot use wildcards, but you can use a blank, which indicates that anonymous users are permitted access.
- **Password:** The password associated with the user account that is being created. You cannot use wildcards, but you can use a blank. If a blank is used, users must supply a blank password when logging on to the server. When you use a GRANT statement to create a password, the password is automatically encrypted when it is saved to the user table.

When you assign a password to a user, you must precede the password with the IDENTIFIED BY subclause. In addition, the password, just like the user and host values, should be enclosed in single quotes.

Now that you have seen how the syntax is defined for the GRANT, ON, and TO clauses, you're ready to take a look at some examples that put all these clauses together.

Creating a Basic GRANT Statement

The first example GRANT statement creates a user account and grants global-level privileges to that account:

```
GRANT ALL
ON *.*
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1';
```

In this statement, the GRANT clause includes one privilege (ALL). The ALL privileges grants all privileges to the user except those that allow the user to grant and revoke privileges. The ON clause includes the double wildcard (*.*) option, which indicates that the privileges being assigned to this user are at the global level. The TO clause specifies the username (user1) and the hostname (domain1.com). The clause also includes the IDENTIFIED BY subclause, which defines the password pw1 on the user account.

> *In order to grant privileges to another user, you must have the necessary privileges. This means that, when you're user account was created, it must have been created with the* GRANT OPTION. *This option is discussed later in the chapter.*

To sum up the GRANT statement, you can say that user1, connecting from domain1.com, can perform all tasks on all tables and databases except for granting and revoking privileges. In addition, user1 must supply the pw1 password when logging on to the MySQL server. If you want to view how this account would be added to the user table (the table associated with global privileges), you can use a SELECT statement similar to the following:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='user1';
```

The statement retrieves only two of the privilege columns, both of which are data-related privileges, as shown in the following results:

```
+-------------+-------+-------------+-------------+
| host        | user  | select_priv | update_priv |
+-------------+-------+-------------+-------------+
| domain1.com | user1 | Y           | Y           |
+-------------+-------+-------------+-------------+
1 row in set (0.00 sec)
```

The next example is far more limited in scope than the previous example. The GRANT statement grants SELECT and UPDATE privileges on all tables in the test database, as shown in the following statement:

```
GRANT SELECT, UPDATE
ON test.*
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1';
```

> *Note that user1 is used as the name in all the example* GRANT *statements in this chapter, but you can assume that, for the purpose of these examples, that a new account is being created when each* GRANT *statement is issued.*

As you can see, the GRANT clause includes the SELECT and UPDATE privileges, the ON clause specifies that the entire test database is affected, and the TO clause identifies user1@domain1.com. A password has also been assigned to this account. The statement, then, allows user1 to connect to the MySQL server from the domain1.com host. The user must supply the pw1 password and can issue only SELECT and UPDATE statements against the tables in the test database.

Because the GRANT statement is specific to a database, you can use the following SELECT statements to see how the user account would be added to the user and db tables:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='user1';
SELECT host, db, user, select_priv, update_priv FROM db WHERE user='user1';
```

The SELECT statements return results similar to the following:

```
+-------------+-------+-------------+-------------+
| host        | user  | select_priv | update_priv |
+-------------+-------+-------------+-------------+
| domain1.com | user1 | N           | N           |
+-------------+-------+-------------+-------------+
1 row in set (0.00 sec)

+-------------+------+-------+-------------+-------------+
| host        | db   | user  | select_priv | update_priv |
+-------------+------+-------+-------------+-------------+
| domain1.com | test | user1 | Y           | Y           |
+-------------+------+-------+-------------+-------------+
1 row in set (0.00 sec)
```

As you can see, user1 is listed in the user table but is assigned no privileges. (Each privilege column has an N as a value.) The user, however, is also listed in the db table, which shows that the user has been assigned privileges on the test table. (Each privilege column has Y as a value.) Next you look at an example that creates table-level privileges. The table used in the example is shown in the following table definition:

```
CREATE TABLE Books
(
   BookID SMALLINT NOT NULL PRIMARY KEY,
   BookTitle VARCHAR(60) NOT NULL,
   Copyright YEAR NOT NULL
)
ENGINE=INNODB;
```

You can assume that the Books table has been added to the test database. Now take a look at a GRANT statement that assigns table-level privileges on the Books table:

```
GRANT SELECT, UPDATE
ON test.Books
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1';
```

As you can see, the example is nearly identical to the previous example GRANT statement; however, the ON clause now specifies a table as well as a database, so the statement is now applied at the table level, rather than the database level. As a result, the user account is added to the tables_priv table rather than the db table. To view how the user account is added to the grant tables, you can use a SELECT statement similar to the following:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='user1';
SELECT host, db, user, table_name, table_priv, column_priv
   FROM tables_priv WHERE user='user1';
```

The statements return results similar to the following:

```
+-------------+-------+-------------+-------------+
| host        | user  | select_priv | update_priv |
+-------------+-------+-------------+-------------+
| domain1.com | user1 | N           | N           |
+-------------+-------+-------------+-------------+
1 row in set (0.00 sec)

+-------------+------+-------+------------+---------------+-------------+
| host        | db   | user  | table_name | table_priv    | column_priv |
+-------------+------+-------+------------+---------------+-------------+
| domain1.com | test | user1 | books      | Select,Update |             |
+-------------+------+-------+------------+---------------+-------------+
1 row in set (0.00 sec)
```

As you can see, the user table is unchanged from the results returned by the previous example, but notice the results returned from the tables_priv table. The table includes the hostname, database name, username, and table name. In addition, the Table_priv column includes the two privilege values (Select and Update) as they were assigned in the `GRANT` statement. As the results indicate, the user can execute `SELECT` and `UPDATE` statements against the Books table in the test database.

Now take a look at an example `GRANT` statement that assigns column-level privileges:

```
GRANT SELECT, UPDATE (BookTitle, Copyright)
ON test.Books
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1';
```

The only difference between this statement and the previous example is that the `UPDATE` privilege includes columns names (BookTitle and Copyright). This means that the user can update only those columns and no other columns in the table. As a result, the user has now been added to the columns_priv table. To demonstrate this, you can use the following `SELECT` statements to retrieve data from the user, tables_priv, and columns_priv tables:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='user1';
SELECT host, db, user, table_name, table_priv, column_priv
   FROM tables_priv WHERE user='user1';
SELECT host, db, user, table_name, column_name, column_priv
   FROM columns_priv WHERE user='user1';
```

The statements should return results similar to the following:

```
+-------------+-------+-------------+-------------+
| host        | user  | select_priv | update_priv |
+-------------+-------+-------------+-------------+
| domain1.com | user1 | N           | N           |
+-------------+-------+-------------+-------------+
1 row in set (0.00 sec)
+-------------+------+-------+------------+------------+-------------+
| host        | db   | user  | table_name | table_priv | column_priv |
+-------------+------+-------+------------+------------+-------------+
| domain1.com | test | user1 | books      | Select     | Update      |
```

```
+-------------+------+-------+------------+------------+-------------+
1 row in set (0.00 sec)

+-------------+------+-------+------------+-------------+-------------+
| host        | db   | user  | table_name | column_name | column_priv |
+-------------+------+-------+------------+-------------+-------------+
| domain1.com | test | user1 | books      | BookTitle   | Update      |
| domain1.com | test | user1 | books      | Copyright   | Update      |
+-------------+------+-------+------------+-------------+-------------+
2 rows in set (0.00 sec)
```

Once again, the results returned from the user table remain unchanged from the previous two examples. Notice that the Column_priv column of the tables_priv table now includes the Update value. (The value has been removed from the Table_priv column.) In addition, two rows have been added to the columns_priv table, one for each column. As you can see, the Update privilege has been assigned to each column, but the user still retains the Select privilege at the table level.

Defining the REQUIRE Clause

The `REQUIRE` clause determines whether a secure connection is required when connecting to the MySQL server and, if so, the degree to which that connection should be secured. Secure connections are implemented in MySQL through the use of the Secure Sockets Layer (SSL) protocol, which is a protocol that uses encryption algorithms to protect data that is sent over unsecured connections, such as the Internet. To make connections even more secure, MySQL supports the use of the x509 standard, an industry standard that relies on the use of certificates to help authenticate the identity of a user trying to connect to a MySQL server. Companies referred to as *certificate authorities* issue the certificates that verify the authenticity of the user trying to establish a connection.

> *To support secure connections between clients and the MySQL server, your system must be configured to support the SSL protocol and the x509 standard, as necessary. A discussion of these topics is beyond the scope of this book. The purpose of this section is merely to demonstrate how to create user accounts that require secure connections. For specific system configuration information, refer to the appropriate product documentation as well as the MySQL Web site (*`www.mysql.com`*).*

To use the `REQUIRE` clause in your `GRANT` statement, you must specify one or more options, as shown in the following syntax:

```
[REQUIRE {NONE | SSL | X509 | {<require definition>}]

<require definition>::=
<require option> [[AND] <require option>] [[AND] <require option>]

<require option>::=
{CIPHER '<string>'}
| {ISSUER '<string>'}
| {SUBJECT '<string>'}
```

As you can see, you can specify `NONE`, `SSL`, `X509`, or one or more of the options available to the `<require definition>` placeholder. The following list describes each of these options:

- **`NONE`:** Indicates that a secure connection is not required. This option is associated with the ssl_type column in the user table. If the `NONE` option is used in the `REQUIRE` clause, a blank value is added to the column, which is the default value. Specifying this option is the equivalent of not specifying a `REQUIRE` clause in your `GRANT` statement.

- **SSL:** Specifies that a secure connection is required, but that it can be any type of SSL connection. This option is associated with the ssl_type column in the user table. If the SSL option is used in the REQUIRE clause, the ANY value is added to the column.
- **X509:** Specifies that a secure connection is required and that the connection requires a valid x509 certificate. This option is associated with the ssl_type column in the user table. If the X509 option is used in the REQUIRE clause, the x509 value is added to the column.
- **CIPHER '<string>':** Specifies the cipher method that should be used for the SSL connection. Cipher methods are based on standards that specify how data is to be encrypted. This option is associated with the ssl_type and ssl_cipher columns in the user table. If the CIPHER '<string>' option is used in the REQUIRE clause, the SPECIFIED value is added to the ssl_type column, and the <string> value is added to the ssl_cipher column.
- **ISSUER '<string>':** Specifies the name of the certificate authority that issues the x509 certificate. The name should be used for an x509 connection. This option is associated with the ssl_type and x509_issuer columns in the user table. If the ISSUER '<string>' option is used in the REQUIRE clause, the SPECIFIED value is added to the ssl_type column, and the <string> value is added to the x509_issuer column.
- **SUBJECT '<string>':** Specifies the subject that should be included on the x509 certificate when establishing a secure connection. This option is associated with the ssl_type and x509_subject columns in the user table. If the SUBJECT '<string>' option is used in the REQUIRE clause, the SPECIFIED value is added to the ssl_type column, and the <string> value is added to the x509_subject column.

If you specify the NONE, SSL, or X509 option, you can specify only one of these options. For the last three options (CIPHER, ISSUER, and STRING), you can specify one or more, in any order. Optionally, if you specify more than one of these three options, you can separate them with the AND keyword. If you don't separate them with the AND keyword, then you simply separate them with a space or line break (no comma). Now take a look at an example of a GRANT statement that uses a REQUIRE clause:

```
GRANT SELECT, UPDATE (BookTitle, Copyright)
ON test.Books
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1'
REQUIRE SSL;
```

As you can see, the REQUIRE clause includes the SSL option, which indicates that an SSL connection is required, but there are no restrictions placed on the nature of the SSL connection. Once you execute the GRANT statement, you can use the following SELECT statement to view the user account information that is added to the user table:

```
SELECT user, ssl_type, ssl_cipher, x509_issuer, x509_subject
FROM user WHERE user='user1';
```

The statement returns results similar to the following:

```
+-------+----------+------------+-------------+--------------+
| user  | ssl_type | ssl_cipher | x509_issuer | x509_subject |
+-------+----------+------------+-------------+--------------+
| user1 | ANY      |            |             |              |
+-------+----------+------------+-------------+--------------+
1 row in set (0.00 sec)
```

As the results show, the encryption-related permission columns are empty except for the ssl_type column, which contains a value of ANY. Now take a look at what happens if you modify the GRANT statement:

```
GRANT SELECT, UPDATE (BookTitle, Copyright)
ON test.Books
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1'
REQUIRE SUBJECT 'test client cert.'
  AND ISSUER 'Test C.A.';
```

Now the REQUIRE clause includes the SUBJECT and ISSUER items. If you run the SELECT statement again, you should see results similar to the following:

```
+-------+-----------+------------+-------------+-------------------+
| user  | ssl_type  | ssl_cipher | x509_issuer | x509_subject      |
+-------+-----------+------------+-------------+-------------------+
| user1 | SPECIFIED |            | Test C.A.   | test client cert. |
+-------+-----------+------------+-------------+-------------------+
1 row in set (0.00 sec)
```

As you can see, the ssl_type column contains the SPECIFIED value, and the x509_issuer and the x509_subject columns include the string values that you specified in the GRANT statement.

Defining the WITH Clause

The last clause in the GRANT statement is the WITH clause, which allows you to specify whether a user can grant and revoke privileges as well as allowing you to specify connection-related privileges, as shown in the following syntax:

```
[WITH <with option> [<with option>...]]

<with option>::=
{GRANT OPTION}
| {MAX_QUERIES_PER_HOUR <count>}
| {MAX_UPDATES_PER_HOUR <count>}
| {MAX_CONNECTIONS_PER_HOUR <count>}
```

As the syntax shows, the WITH clause can include from one to four options. The following list describes each of these options:

- **GRANT OPTION:** Specifies that users can execute GRANT, REVOKE, and DROP USER statements. Using this option in this clause has the same effect as using the option in the GRANT clause. Grant privileges are applied to the user at the level that the GRANT statement is applied (as defined in the ON clause) and to the limit that the grantor is permitted access to the system. This option sets the Grant_priv column in the user, db, or host table or adds the Grant value to the Table_priv column of the tables_priv table. (Using the GRANT OPTION in the WITH clause achieves the same results as using the GRANT OPTION as a privilege in the GRANT clause.)
- **MAX_QUERIES_PER_HOUR <count>:** The number of queries permitted in an hour. The number is added to the max_questions column of the user table. If the option is not specified, the default value is 0, which indicates that an unlimited number of queries are permitted.

- **MAX_UPDATES_PER_HOUR <count>:** The number of data modifications permitted in an hour. The number is added to the max_updates column of the user table. If the option is not specified, the default value is 0, which indicates that an unlimited number of updates are permitted.
- **MAX_CONNECTIONS_PER_HOUR <count>:** The number of connections permitted in an hour. The number is added to the max_connections column of the user table. If the option is not specified, the default value is 0, which indicates that an unlimited number of connections are permitted.

When you create a GRANT statement that includes the WITH clause, you simply specify the options and, if appropriate, specify a value. You do not need to separate the options with a comma. For example, the following GRANT statement specifies the MAX_QUERIES_PER_HOUR and MAX_UPDATES_PER_HOUR options, setting the value of each to 50:

```
GRANT SELECT, UPDATE
ON test.*
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1'
WITH GRANT OPTION MAX_QUERIES_PER_HOUR 50 MAX_UPDATES_PER_HOUR 50;
```

As the statement shows, the connection-related values are added to the applicable columns in the user table. The other privileges are assigned at the database level. To verify this, you can run the following SELECT statements:

```
SELECT user, grant_priv, max_questions, max_updates, max_connections
   FROM user WHERE user='user1';
SELECT user, grant_priv FROM db WHERE user='user1';
```

The statements return results similar to the following:

```
+-------+------------+---------------+-------------+-----------------+
| user  | grant_priv | max_questions | max_updates | max_connections |
+-------+------------+---------------+-------------+-----------------+
| user1 | N          |            50 |          50 |               0 |
+-------+------------+---------------+-------------+-----------------+
1 row in set (0.00 sec)

+-------+------------+
| user  | grant_priv |
+-------+------------+
| user1 | Y          |
+-------+------------+
1 row in set (0.00 sec)
```

As you can see, the account as it appears in the user table has not been granted the GRANT OPTION privilege, but the max_questions and max_updates columns each contain 50. If the user tries to exceed this number of queries or updates, the operations are not allowed. The results returned by the SELECT statement also show that the user has been added to the db table, where the GRANT OPTION privilege has been assigned.

Once you add a user to your system, it is often useful to view what privileges have been assigned to that user. This is helpful in troubleshooting connections and in changing privileges — or revoking them. To view the privileges that have been assigned to a user, you can use SELECT statements to retrieve all columns for all grant tables, which is a very cumbersome process. However, you can also use the SHOW GRANTS statement, which is a far more efficient method to use to display a user's privileges.

Using the SHOW GRANTS Statement

The SHOW GRANTS statement displays information about a specific user account. To use the statement, you must include the SHOW GRANTS FOR keywords, followed by the username and hostname, as shown in the following syntax:

```
SHOW GRANTS FOR '<user>'@'<host>'
```

As you can see, the syntax for a SHOW GRANTS statement is very basic. You want to be sure that, when specifying the username and hostname, you enclose each name in single quotes and you separate the names with the at symbol (@). For example, the following SHOW GRANTS statement displays the user account information for user1@domain1.com:

```
SHOW GRANTS FOR 'user1'@'domain1.com';
```

When you execute this statement, you should receive results similar to the following:

```
+-----------------------------------------------------------------------------------+
| Grants for user1@domain1.com                                                      |
+-----------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'user1'@'domain1.com' IDENTIFIED BY PASSWORD
'*2B602296A79E0A8784ACC5C88D92E46588CCA3C3' WITH MAX_QUERIES_PER_HOUR 50
MAX_UPDATES_PER_HOUR 50                                                           |
| GRANT SELECT, UPDATE ON 'test'.* TO 'user1'@'domain1.com' WITH GRANT OPTION     |
+-----------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

The first row returned by the SHOW GRANTS statements displays information from the user table. In the previous results, this row is broken over three lines, but what you actually see depends on your system. In this case, the row begins with the GRANT USAGE ON keywords. Usage is a privilege option that indicates that no privileges have been set for that account. Whenever a user account has not been assigned global data-related or administrative privileges, the USAGE option is used. If global privileges have been granted, those privilege options are displayed. The first row also displays the username, hostname, and encrypted password. If encryption-related or connection-related privileges have been assigned to an account, they're displayed after the username, hostname, and password. For this user, the MAX_QUERIES_PER_HOUR and MAX_UPDATES_PER_HOUR privileges are displayed, which are associated with the max_questions and max_updates columns, respectively.

If the SHOW GRANTS statement returns more than one row, those additional rows display details about nonglobal privileges. For example, the second row returned by the preceding SHOW GRANTS statement indicates that the SELECT and UPDATE privileges have been granted on the test database (test.*) and that the GRANT OPTION has been assigned at the database level. If table or column level privileges had been assigned to this user, they would also be returned by the SHOW GRANTS statement.

Now that you have seen how to add users to your MySQL environment, configure privileges for those users, and display user account information, you're ready to try it out for yourself. In the following exercise, you use the GRANT statement to add users to the DVDRentals database and assign privileges to those users, and you use the SHOW GRANTS statement to verify that those privileges have been correctly assigned.

Try It Out Adding Users to the DVDRentals Database

The following steps describe how to add users to the DVDRentals database and assign privileges to those users:

1. Open the mysql client utility. If the mysql database is not the active database, type the following command, and press Enter:

```
use mysql
```

   You should receive a message indicating that you switched to the mysql database.

2. The first user account that you create is named myuser1, which is allowed to connect only from the local computer. Execute the following SQL statement at the mysql command prompt:

```
GRANT SELECT, UPDATE (EmpFN, EmpMN, EmpLN)
ON DVDRentals.Employees
TO 'myuser1'@'localhost' IDENTIFIED BY 'mypw1'
WITH MAX_CONNECTIONS_PER_HOUR 25;
```

   You should receive a message indicating that the statement successfully executed.

3. When you created the myuser1 user account, the user should have been added to the user, tables_priv, and columns_priv tables. Retrieve data from those tables to verify that the user has been added. Execute the following SQL statements at the mysql command prompt:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='myuser1';
SELECT host, db, user, table_name, table_priv, column_priv
   FROM tables_priv WHERE user='myuser1';
SELECT host, db, user, table_name, column_name, column_priv
   FROM columns_priv WHERE user='myuser1';
```

   You should receive results similar to the following:

```
+-----------+---------+-------------+-------------+
| host      | user    | select_priv | update_priv |
+-----------+---------+-------------+-------------+
| localhost | myuser1 | N           | N           |
+-----------+---------+-------------+-------------+
1 row in set (0.00 sec)

+-----------+------------+---------+------------+------------+-------------+
| host      | db         | user    | table_name | table_priv | column_priv |
+-----------+------------+---------+------------+------------+-------------+
| localhost | dvdrentals | myuser1 | employees  | Select     | Update      |
+-----------+------------+---------+------------+------------+-------------+
1 row in set (0.00 sec)

+-----------+------------+---------+------------+-------------+-------------+
| host      | db         | user    | table_name | column_name | column_priv |
+-----------+------------+---------+------------+-------------+-------------+
| localhost | dvdrentals | myuser1 | employees  | EmpMN       | Update      |
| localhost | dvdrentals | myuser1 | employees  | EmpLN       | Update      |
| localhost | dvdrentals | myuser1 | employees  | EmpFN       | Update      |
+-----------+------------+---------+------------+-------------+-------------+
3 rows in set (0.00 sec)
```

4. Now use the `SHOW GRANTS` statement to display the privileges assigned to the myuser1 user account. Execute the following SQL statement at the mysql command prompt:

```
SHOW GRANTS FOR 'myuser1'@'localhost';
```

You should receive results similar to the following:

```
+-----------------------------------------------------------------------------------+
| Grants for myuser1@localhost                                                      |
+-----------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'myuser1'@'localhost' IDENTIFIED BY PASSWORD
'*129A95F81EAFD64723D26F1872A8F27B22A25B48' WITH MAX_CONNECTIONS_PER_HOUR 25       |
| GRANT SELECT, UPDATE (EmpFN, EmpLN, EmpMN) ON `dvdrentals`.`employees` TO
'myuser1'@'localhost'                                                             |
+-----------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

5. To verify that the myuser1 user account has been properly added to your system, exit the mysql client utility and then relaunch the utility as the new user. Close the mysql client utility by executing the following command:

```
exit
```

You should be returned to your operating system's command prompt.

6. Now execute the following command at your operating system's command prompt:

```
mysql -u myuser1 -p DVDRentals
```

When prompted, type the password mypw1, and then press Enter. You should now be at the mysql command prompt.

7. Next, try to retrieve data from the Employees table. Execute the following SQL statements at the mysql command prompt:

```
SELECT * FROM Employees;
```

You should receive results similar to the following:

```
+-------+--------+-------+-----------+
| EmpID | EmpFN  | EmpMN | EmpLN     |
+-------+--------+-------+-----------+
|     1 | John   | P.    | Smith     |
|     2 | Robert | NULL  | Schroader |
|     3 | Mary   | Marie | Michaels  |
|     4 | John   | NULL  | Laguci    |
|     5 | Rita   | C.    | Carter    |
|     6 | George | NULL  | Brooks    |
+-------+--------+-------+-----------+
6 rows in set (0.04 sec)
```

8. Now try to insert data in the Employees table. Execute the following SQL statements at the mysql command prompt:

```
INSERT INTO Employees VALUES ('Sarah', 'Louise', 'Peterson');
```

You should receive an error message indicating that you cannot perform this operation.

9. Close the mysql client utility by executing the following command:

```
exit
```

You should be returned to your operating system's command prompt.

10. Log back in the mysql client utility as the root user, with the mysql database as the active database.

11. Next, create the mysqlapp user account, which is permitted to access the MySQL server from the local computer. Execute the following SQL statement at the mysql command prompt:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON DVDRentals.*
TO 'mysqlapp'@'localhost' IDENTIFIED BY 'pw1';
```

You should receive a message indicating that the statement successfully executed.

12. To verify that the mysqlapp user account has been added to the user and db tables, execute the following SQL statement at the mysql command prompt:

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='mysqlapp';
SELECT host, db, user, select_priv, update_priv FROM db WHERE user='mysqlapp';
```

You should receive results similar to the following:

```
+-----------+----------+-------------+-------------+
| host      | user     | select_priv | update_priv |
+-----------+----------+-------------+-------------+
| localhost | mysqlapp | N           | N           |
+-----------+----------+-------------+-------------+
1 row in set (0.00 sec)

+-----------+------------+----------+-------------+-------------+
| host      | db         | user     | select_priv | update_priv |
+-----------+------------+----------+-------------+-------------+
| localhost | dvdrentals | mysqlapp | Y           | Y           |
+-----------+------------+----------+-------------+-------------+
1 row in set (0.03 sec)
```

13. Now display the privileges that are assigned to the mysqlapp user account. Execute the following SQL statement at the mysql command prompt:

```
SHOW GRANTS FOR 'mysqlapp'@'localhost';
```

You should receive results similar to the following:

```
+-----------------------------------------------------------------------------------+
| Grants for mysqlapp@localhost                                                     |
+-----------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'mysqlapp'@'localhost' IDENTIFIED BY PASSWORD
'*2B602296A79E0A8784ACC5C88D92E46588CCA3C3'                                        |
| GRANT SELECT, INSERT, UPDATE, DELETE ON 'dvdrentals'.* TO 'mysqlapp'@'localhost'|
+-----------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

14. To verify that the user account can connect to the MySQL server, you must exit the mysql client utility and then relaunch the utility as the new user. Close the mysql client utility by executing the following command:

```
exit
```

You should be returned to your operating system's command prompt.

15. Now execute the following command at your operating system's command prompt:

```
mysql DVDRentals -u mysqlapp -p
```

When prompted, type the password pw1, and then press Enter. You should now be at the mysql command prompt.

16. Close the mysql client utility by executing the following command:

```
exit
```

You should be returned to your operating system's command prompt.

How It Works

To create the first user account in this exercise, you used the following GRANT statement:

```
GRANT SELECT, UPDATE (EmpFN, EmpMN, EmpLN)
ON DVDRentals.Employees
TO 'myuser1'@'localhost' IDENTIFIED BY 'mypw1'
WITH MAX_CONNECTIONS_PER_HOUR 25;
```

The first clause in this statement is the GRANT clause, which grants the SELECT privilege and the UPDATE privilege. The UPDATE privilege is specific to the EmpFN, EmpMN, and EmpLN columns of the Employees table. The ON clause specifies the database name (DVDRentals) and the table (Employees). As a result, the privileges apply only to the Employees table. The TO clause specifies the name of the user (myuser1) and the host from which the user can connect to the MySQL server (localhost). The TO clause also includes an IDENTIFIED BY clause, which specifies that the password mypw1 will be assigned to the account. The GRANT statement also includes a WITH clause that contains the MAX_CONNECTIONS_PER_HOUR option. The user account is permitted up to 25 connections per hour.

After you executed the GRANT statement, you executed three SELECT statements that retrieved data from the user, tables_priv, and columns_priv tables. The first SELECT statement retrieved the values in the Host, User, Select_priv, and Update_priv columns of the user table. The results showed that a user named myuser1 had been created and that the account could log on to the MySQL server from the local computer. The results also showed that the user did not have SELECT or UPDATE privileges at the global level, which is to be expected because the privileges are assigned at a table and column level.

The second SELECT statement retrieved data from the Host, Db, User, Table_name, Table_priv, and Column_priv columns of the tables_priv table. As you would expect, the user account and host are listed. In addition, the account is assigned SELECT table privileges and UPDATE column privileges on the Employees table of the DVDRentals database. Because the Column_priv column includes the Update value, you would expect the columns_priv table to contain the applicable privilege for each column specified in the GRANT clause of the GRANT statement.

To verify this, the third SELECT statement retrieved the Host, Db, User, Table_name, Column_name, and Column_priv columns from the columns_priv table. The results show that a row has been added to the table for each column specified in the GRANT statement, indicating that the user can update the EmpFN, EmpMN, and EmpLN columns of the Employees table.

You also used the SHOW GRANTS statement to confirm that the new user account was assigned the appropriate privileges:

```
SHOW GRANTS FOR 'myuser1'@'localhost';
```

The statement specifies the user account name and the host from which the user account can connect. When you executed the statement, you received a result set that contained one column and two rows. The first row shows the global settings and privileges that are assigned to the user account. In this case, the user can log on to the MySQL server from the localhost. The user must supply a password and cannot initiate more than 25 connections in an hour. The user, though, is not assigned any global privileges, as indicated by the USAGE privilege.

The second row of the results returned by the SHOW GRANTS statement indicates that the SELECT privilege and UPDATE privilege have been assigned to the user for the Employees table of the DVDRentals database. The UPDATE privilege applies only to the specified columns in the Employees table.

After you viewed the privileges assigned to the myuser1 account, you exited the MySQL client utility and then relaunched the utility as the new user. You then tried to execute a SELECT statement against the Employees table and then an INSERT statement. As you would expect, you were able to retrieve data from the table, but not add data to the table. However, if you had tried to update any part of an employee's name, you would have been permitted to do so because the UPDATE privilege was also assigned to this user account.

After you tested the connection for the new user account, you exited mysql and then logged back in as the root user. From there, you executed the following GRANT statement:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON DVDRentals.*
TO 'mysqlapp'@'localhost' IDENTIFIED BY 'pw1';
```

The GRANT statement assigns the SELECT, INSERT, UPDATE, and DELETE privileges for all tables in the DVDRentals database. The statement also creates a user account named mysqlapp. The user can connect only from the local computer. To connect to the MySQL server, the user must supply the pw1 password.

After you executed the GRANT statement, you used two SELECT statements to verify that the user and db tables contained the correct data. As you would expect, the user table includes the account, but no privileges have been assigned because the privileges are assigned at a database level, not a global level. The second SELECT statement verifies this. The data returned by that statement shows that privileges have been granted to that user on the DVDRentals database. Although only two of the four privileges were returned by the SELECT statement, you were able to verify that all privileges had been granted to the account by running the following SHOW GRANTS statement:

```
SHOW GRANTS FOR 'mysqlapp'@'localhost';
```

The results returned by the statement indicate that the user can log on to the MySQL server, that no global privileges have been granted, and that the SELECT, INSERT, UPDATE, and DELETE privileges have been granted to this account.

Setting Passwords for MySQL User Accounts

There might be times when, after a user account has been created, you want to add a password to the user account or change the existing password. MySQL provides the SET PASSWORD statement for setting a user account password; however, setting the new password doesn't mean that the password is immediately implemented. After setting a new password, you should use the FLUSH PRIVILEGES statement to reload the grant tables in your system's memory to ensure that the most current user information (including the new password) is being used.

Using the SET PASSWORD Statement

The SET PASSWORD statement can be used to set the password of another user account or of your own user account. The syntax for the SET PASSWORD statement is as follows:

```
SET PASSWORD [FOR '<user>'@'<host>'] = PASSWORD('<new password>')
```

As you can see, you must specify the SET PASSWORD keywords, the equal sign, and the PASSWORD() function, using the new password as an argument in the function. You can also include the optional FOR clause in your statement. If you're setting the password for the user account under which you're currently logged on to the system, you don't need to use the FOR clause. For example, the following statement allows you to change your password to pw2:

```
SET PASSWORD = PASSWORD('pw2');
```

As you can see, you have to specify only the required statement elements to change your own password. Notice that the new password is included as an argument in the PASSWORD() function and that the new password is enclosed in single quotes. If you are setting a password for a user account other than the one you used to log on to the system, you can use a statement similar to the following:

```
SET PASSWORD FOR 'user1'@'domain1.com' = PASSWORD('pw3');
```

The statement includes a FOR clause, which identifies the user account as user1@domain1.com. All other aspects of the statement are the same as the previous example (except that the new password is now pw3). Once you set a password for an account, it takes effect when you restart the MySQL server or you flush the privileges, which means that you reload the grant tables in your system's memory. To flush the privileges, you can use the FLUSH PRIVILEGES statement.

Using the FLUSH PRIVILEGES Statement

The FLUSH PRIVILEGES statement includes no options or clauses other than the FLUSH PRIVILEGES keywords. As a result, to reload the grant tables into memory, all you need to do is execute the following statement:

```
FLUSH PRIVILEGES;
```

As soon as you execute this statement, the grant tables are reloaded and any new user account information is put into effect.

Now that you know how to assign a password to a user account and reload the grant tables in your system's memory, you're ready to try out the `SET PASSWORD` and `FLUSH PRIVILEGES` statements. In the following exercise you change the password of the myuser1 account that you created in the last Try It Out section.

Try It Out Changing a User Account Password

The following steps describe how to change the password for the myuser1 account:

1. Open the mysql client utility as the root user.
2. To change the password of the myuser1 account, execute the following SQL statement at the mysql command prompt:

```
SET PASSWORD FOR 'myuser1'@'localhost' = PASSWORD('mypw2');
```

   You should receive a message indicating that the statement successfully executed.

3. Once you change the password, you should reload the grant tables in your system's memory to ensure that the new password takes effect. Execute the following SQL statement at the mysql command prompt:

```
FLUSH PRIVILEGES;
```

   You should receive a message indicating that the statement successfully executed.

4. Next, try out the new password by exiting the mysql client utility and then relaunching the utility as the myuser1 account. Close the mysql client utility by executing the following command:

```
exit
```

   You should be returned to your operating system's command prompt.

5. Execute the following SQL statement at the mysql command prompt:

```
mysql DVDRentals -u myuser1 -p
```

   When prompted, type the password mypw2, and then press Enter. You should now be at the mysql command prompt.

6. Close the mysql client utility by executing the following command:

```
exit
```

   You should be returned to your operating system's command prompt.

How It Works

To set the password for the muuser1 account, you used the following `SET PASSWORD` statement:

```
SET PASSWORD FOR 'myuser1'@'localhost' = PASSWORD('mypw2');
```

The statement specifies the `SET PASSWORD FOR` keywords, followed by the name of the user account (`myuser1`) and the host (`localhost`) from which the user is permitted to connect to the MySQL server. The statement then uses the `PASSWORD()` function to encrypt and set the new password (`mypw2`).

Because a password already existed for the user account, the new password simply replaced the old password in the user table. If a password had not existed for that account, the new password would have been added to the user table.

Once you assigned the new password to the myuser1 account, you used the following statement to reload the grant tables in your system's memory:

```
FLUSH PRIVILEGES;
```

The `FLUSH PRIVILEGES` statement ensures that the most current user account information is applied when a user logs into the MySQL server.

Dropping Users and Revoking Privileges

Quite often you will find that, once you've created a user account, you want to remove that account from your system or modify the account's privileges. Removing an account often includes three steps:

1. Using the `SHOW GRANTS` statement to view the user account's current privileges.
2. Using the `REVOKE` statement to revoke the privileges from the user account.
3. Using the `DROP USER` statement to remove the user from the system.

All privileges must have been revoked from a user account before you can use the `DROP USER` statement to remove the user; however, you do not always need to take each step described here. For example, if you know what privileges have been granted to the user, you don't have to use the `SHOW GRANTS` statement to view those privileges. In addition, if you know that no privileges have been granted to the user, you do not have to use the `REVOKE` statement before using the `DROP USER` statement. You might also find that you want to revoke certain privileges but not remove the user from the system. In that case, you need to use only the `SHOW GRANTS` statement and the `REVOKE` statement.

Now that you have an overview of how the user account modification and removal process works, you can take a close look at the statement used to modify the accounts. You've already seen how you can use the `SHOW GRANTS` statement to display privilege information. Now take a look at the `REVOKE` and `DROP USER` statements.

Using the REVOKE Statement

The `REVOKE` statement allows you to remove privileges for a user account. When you revoke privileges, the user account record is removed from db, host, tables_priv, and columns_priv tables, whichever are applicable, and all privileges are revoked from the user table. But the user account is still listed in the user table. (To remove the account from the user table, you must use the `DROP USER` statement, which is described in the text that follows.)

Before you can actually drop the user account from the user table, you should use the `REVOKE` statement to revoke all privileges. The `REVOKE` statement takes two forms. The first form removes all privileges from the user account. This is the simplest method to use if you're simply going to drop the user from the system and are not going to revoke only specific privileges. The following syntax describes how the first form of the `REVOKE` statement is defined:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM '<user>'@'<host>' [{, '<user>'@'<host>'}...]
```

As you can see, you must specify the REVOKE clause and the FROM clause. The FROM clause identifies one or more user accounts. To illustrate how the REVOKE statement works, take a look at an example based on a user account created with the following GRANT statement:

```
GRANT SELECT, UPDATE
ON test.*
TO 'user1'@'domain1.com' IDENTIFIED BY 'pw1'
WITH GRANT OPTION MAX_QUERIES_PER_HOUR 50 MAX_UPDATES_PER_HOUR 50;
```

The user1 account has been granted SELECT and UPDATE privileges on the test database. The account has also been assigned two connection-related privileges. To use the first form of the REVOKE statement to remove this account, you can use the following statement:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'user1'@'domain1.com';
```

Notice that the statement includes only the REVOKE clause and the FROM clause, which identifies the user1@domain1.com user account. You can also revoke the privileges from the user1 account by using a long form of the REVOKE statement, which is shown in the following syntax:

```
REVOKE <privilege> [(<column> [{, <column>}...])]
   [{, <privilege> [(<column> [{, <column>}...])]}...]
ON {<table> | * | *.* | <database>.*}
FROM '<user>'@'<host>' [{, '<user>'@'<host>'}...]
```

As you can see, the REVOKE clause now includes <privilege> and <column> placeholders. The clause is defined exactly as it is defined in the GRANT clause of the GRANT statement. You must specify each privilege that you want to revoke. If privileges are assigned at the column level, then you must also identify those columns. The ON clause of the long version of the REVOKE statement is also the same as the ON clause of the GRANT statement. The same options are available, and the option you pick depends on the scope of the privileges.

The long form of the REVOKE statement also includes a FROM clause, which works the same as the FROM clause in the short form of the statement. For example, if you want to use the long form of the statement to revoke the privileges of the user1 account, you would use the following REVOKE statement:

```
REVOKE SELECT, UPDATE, GRANT OPTION
ON test.*
FROM 'user1'@'domain1.com';
```

As you can see, the REVOKE clause includes all the privileges that have been granted to the user account (including the GRANT OPTION privilege), the ON clause specifies the test database, and the FROM clause identifies the user account.

The primary advantage of the long form of the REVOKE statement over the short form is that the long form allows you to revoke all privileges or only certain privileges, whereas the short form allows you to revoke all privileges only.

When you revoke all the privileges from a user account, that account is removed from all grant tables except the user table. In addition, any privileges that had been set in the user table are also revoked. To remove the user account from the user table after all privileges have been revoked, you must use the `DROP USER` statement.

> *The* `REVOKE` *statement does not remove a user account from grant tables other than the user table if that user account is not listed in the user table. For example, if the db table includes a user account that is not in the user table, the* `REVOKE` *statement does not remove the user from the db table. In a case like this, you must use a* `DELETE` *statement to remove that row from the table.*

Using the DROP USER Statement

The `DROP USER` statement requires only the `DROP USER` keywords and one or more user accounts, as shown in the following syntax:

```
DROP USER '<user>'@'<host>' [{, '<user>'@'<host>'}...]
```

If you specify multiple accounts, you must separate them by a comma. Remember, to drop the accounts, there can be no privileges assigned to the account. If you try to execute a `DROP USER` statement and it fails, use the `SHOW GRANTS` statement to determine whether any privileges still exist for that account.

Now take a look at an example of a `DROP USER` statement. The following statement shows how you would drop the user1@domain1.com user account:

```
DROP USER 'user1'@'domain1.com';
```

As you can see, the statement includes only the `DROP USER` keywords, along with the username and hostname, each enclosed in single quotes and separated by the at (@) symbol.

Now that you've learned how to revoke privileges and drop users from your system, you're ready to try it out for yourself. In the following exercise, you revoke the privileges of the myuser1 account that you created in an earlier Try It Out section, and then you remove this account from your system.

Try It Out — Removing Users from the DVDRentals Database

The following steps describe how to revoke the privileges of the myuser1 account and then drop that user from your system:

1. Open the mysql client utility. If the mysql database is not the active database, execute the following command:

```
use mysql
```

   You should receive a message indicating that you switched to the DVDRentals database.

2. Before revoking the privileges of the myuser1 account, you should view the current privileges. Execute the following SQL statement at the mysql command prompt:

```
SHOW GRANTS FOR 'myuser1'@'localhost';
```

You should receive results similar to the following:

```
+-----------------------------------------------------------------------------------+
| Grants for myuser1@localhost                                                      |
+-----------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'myuser1'@'localhost' IDENTIFIED BY PASSWORD
'*B7F1931E8F564F69038DB9132AB2ED7F144D8414' WITH MAX_CONNECTIONS_PER_HOUR 25      |
| GRANT SELECT, UPDATE (EmpFN, EmpMN, EmpLN) ON `dvdrentals`.`employees` TO
'myuser1'@'localhost'                                                             |
+-----------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

3. You can use the information returned by the `SHOW GRANTS` statement to create a `REVOKE` statement. Execute the following SQL statement at the mysql command prompt:

```
REVOKE SELECT, UPDATE (EmpFN, EmpMN, EmpLN)
ON DVDRentals.Employees
FROM 'myuser1'@'localhost';
```

You should receive a message indicating that the statement successfully executed.

4. Once you have revoked all the permissions from the myuser1 account, you should drop the user from your system. Execute the following SQL statement at the mysql command prompt:

```
DROP USER 'myuser1'@'localhost';
```

You should receive a message indicating that the statement successfully executed.

5. Now that you have learned how to remove users from your system, you should remove any anonymous users that are created when you installed MySQL. For Linux, you should remove the anonymous users from the user table. For Windows and Linux, you should remove the anonymous users in the db table. If you're working in a Windows environment, skip ahead to Step 7. If you're working in a Linux environment, execute the following SQL statement at the mysql command prompt:

```
DROP USER ''@'localhost';
```

You should receive a message indicating that the statement successfully executed.

6. If you're working in a Linux environment, execute the following SQL statement at the mysql command prompt:

```
DROP USER ''@'<host>';
```

The *<host>* placeholder refers to the name of the local computer. Once you execute the statement, you should receive a message indicating that the statement successfully executed.

7. For both Windows and Linux environments, execute the following SQL statement at the mysql command prompt:

```
DELETE FROM db WHERE user='';
```

You should receive a message indicating that the statement successfully executed and that two rows were affected.

8. Exit the mysql client utility.

How It Works

The first step that you took in deleting the myuser1 account from your system was to execute the following SHOW GRANTS table:

```
SHOW GRANTS FOR 'myuser1'@'localhost';
```

The results returned by the statement indicated that the myuser1 account had been granted no global data-related or administrative privileges but was configured with the MAX_CONNECTIONS_PER_HOUR option set to 25. In addition, the user was granted SELECT and UPDATE privileges on the Employees table of the DVDRentals database. The user, though, could update only the EmpFN, EmpMN, and EmpLN columns of the Employees table.

From the information returned by the SHOW GRANTS statement, you were able to create a REVOKE statement that revokes the applicable permissions:

```
REVOKE SELECT, UPDATE (EmpFN, EmpMN, EmpLN)
ON DVDRentals.Employees
FROM 'myuser1'@'localhost';
```

In this statement, the REVOKE clause specifies SELECT and UPDATE privileges. The statement specifies the UPDATE privilege on the EmpFN, EmpMN, and EmpLN columns only. The ON clause specifies that the privileges apply to the Employees table of the DVDRentals database, and the FROM clause specifies that the privileges be revoked from the myuser1 account associated with the local host.

This exercise used the long form of the REVOKE *statement to revoke the privileges in order to demonstrate how the more complicated form of the statement works. You could have also used the short form, in which case, you could have used the following statement:*

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'myuser1'@'localhost';
```

When you executed the REVOKE statement, rows related to the user were removed from the tables_priv and columns_priv tables. In addition, any privileges granted in the user table were revoked. You then used the following statement to remove the user from your system:

```
DROP USER 'myuser1'@'localhost';
```

The statement includes the DROP USER keywords, the name of the user (myuser1), and the name of the host (localhost). When you executed this statement, the myuser1 account was removed from the user table.

Once you dropped the myuser1 account from your system, you dropped anonymous user accounts from your system. These are the user accounts that have a blank value in the User column. By dropping the anonymous accounts, you are preventing anonymous users from being able to log on to the MySQL server. Even if no privileges are assigned to an account, any account that is listed in the user table can log on to the server. In addition, because anonymous user accounts were added to the user table only for the Linux installation, you had to use the DROP USER statements only for the Linux environment. Note, however, that you did not have to revoke permissions before deleting the accounts because no privileges had been assigned to the anonymous accounts.

In addition to anonymous accounts existing in the user table of a Linux installation, the db table for both Linux and Windows installations contained anonymous accounts. Because these accounts have no accounts associated with them in the user table, you cannot use the `REVOKE` statement or the `DROP USER` statement on the account, but instead must use a `DELETE` statement to remove the users. To drop the anonymous users from the db table in your installation, you created a `DELETE` statement whose `WHERE` clause specified a blank user (`WHERE user=''`). When you executed the statement, the user accounts were removed from the table. The only user account that should now be listed in your db table is the mysqlapp account, which you created in an earlier Try It Out section.

Summary

As the chapter has demonstrated, MySQL allows you to control who has access to the MySQL server and allows you to assign privileges to individual user accounts. You can assign these privileges at a global level, database level, table level, or column level, and you can mix the level for individual users. For example, you can allow a user to view data from all tables in a database, permit the user to add data to only one table in the database, and restrict the user to being able to update only one column in that table. To support your ability to administer MySQL security, this chapter covered the following topics:

- Understanding how the user, db, host, tables_priv, and columns_priv tables provide security
- Learning how users are authenticated to access the MySQL server and authorized to perform specific operations
- Using `GRANT` statements to create user accounts and assign privileges
- Using `SHOW GRANTS` statements to display user privileges
- Using `SET PASSWORD` statements to assign or change user account passwords
- Using `FLUSH PRIVILEGES` statements to reload the grant tables in your system's memory
- Using `REVOKE` statements to revoke privileges assigned to user accounts
- Using `DROP USER` statements to remove a user account from MySQL

Although MySQL provides an effective infrastructure for protecting the data in the databases, your security strategies should not be limited to the MySQL environment alone. Be sure to implement the security necessary to protect the MySQL-related files stored in your directories so that no unauthorized users can access or copy those files. Also be sure that your network is protected from unauthorized access. To carry out these measures, you should work closely with your system and network administrators and consult the applicable documentation. For current information on MySQL security, be sure to consult the MySQL Web site (`www.mysql.com`). You should take every step necessary to ensure the security of your MySQL installation before you implement MySQL in a production environment. Once you're confident that your system is secure, you can start looking at ways to optimize your system's performance. In Chapter 15, you learn how you can improve the performance of your queries so that you can retrieve and modify data as efficiently as possible.

Exercises

In this chapter, you learned how to create user accounts, assign privileges to those accounts, revoke those privileges, and remove the user accounts from your system. The following exercises help you build on your ability to perform these tasks. The exercises are based on the following database and table definitions:

```
CREATE DATABASE Books;
use Books;
CREATE TABLE Publishers
(
   PubID INT PRIMARY KEY,
   PubName VARCHAR(40) NOT NULL,
   City VARCHAR(20)
);
```

To view solutions to these exercises, see Appendix A.

1. Create a user account for a user named mgr1. The user account should be able to connect from the local computer, using the password mgr1pw. Grant the user account `SELECT` and `INSERT` privileges on the Publishers table. In addition, grant the user `UPDATE` privileges on the PubName and City columns of the Publishers table.
2. Create an SQL statement that changes the mgr1 user account password to mgr2pw.
3. Create an SQL statement that flushes the grant tables and reloads them in your system's memory.
4. Create an SQL statement that displays the privileges assigned to the mgr1 user account.
5. Create an SQL statement that revokes the privileges assigned to the mgr1 user account.
6. Create an SQL statement that removes the mgr1 user account from your system.